

Regular realizability problems and regular languages

A. Rubtsov*

¹ Moscow Institute of Physics and Technology

² National Research University Higher School of Economics
rubtsov99@gmail.com

We investigate regular realizability (RR) problems, which are the problems of verifying whether intersection of a regular language – the input of the problem – and fixed language called filter is non-empty. We consider two kind of problems depending on representation of regular language. If a regular language on input is represented by a DFA, then we obtain (deterministic) regular realizability problem and we show that in this case the complexity of regular realizability problem for an arbitrary regular filter is either **L**-complete or **NL**-complete. We also show that in case of representation regular language on input by NFA the problem is always **NL**-complete.

1 Introduction

The regular realizability problems are the problems of verifying whether intersection of a regular language – the input of the problem – and fixed language called filter is non-empty. Filter F is a parameter of the problem. Depending on representation of a regular language we distinguish the deterministic RR problems $RR(F)$ and the nondeterministic ones $RR^n(F)$, which are corresponds to the description of the regular language either by a deterministic or by a nondeterministic finite automaton.

The main question of studying regular realizability problems is the investigation of it's algorithmic complexity depending on a filter. Algorithmic complexity of corresponding regular realizability problem is a kind of a complexity measure on languages. Investigation of the problems algorithmic complexity in case of regular filters is a natural question. Moreover, the relation between algorithmic complexities of $RR(F)$ and $RR^n(F)$ is still unknown, but only in the case of regular filters we know the separation modulo $L \neq \mathbf{NL}$ conjecture. Our main result is the separation of regular languages into two classes: the first class contains languages with correspondent deterministic RR-problems belong to class **L** and the second class contains languages with correspondent **NL**-complete deterministic RR-problems.

Deterministic regular realizability problems corresponds to a computational model, called Generalized Nondeterministic Automata (GNA). We investigate this model in section 3.

* Supported in part by RFBR grant 14-01-00641.

2 RR-problems and deterministic finite state transductions

In this section we define RR-problems formally and show how its algorithmic complexity relates to deterministic finite state transductions on filters. We study the deterministic version ($\text{RR}(F)$) as the main one, because as we show further the nondeterministic version is too powerful for regular filters – for all nonempty regular languages corresponding RR^n -problems are **NL**-complete.

Definition 1. The regular realizability problem $\text{RR}(F)$ is the problem of verifying non-emptiness of the intersection of the filter F with a regular language $L(\mathcal{A})$, where \mathcal{A} is a DFA. Formally

$$\text{RR}(F) = \{\mathcal{A} \mid \mathcal{A} \text{ is DFA and } L(\mathcal{A}) \cap F \neq \emptyset\}.$$

In the same way we define the nondeterministic version:

$$\text{RR}^n(F) = \{\mathcal{A} \mid \mathcal{A} \text{ is NFA and } L(\mathcal{A}) \cap F \neq \emptyset\}.$$

Since we are going to consider classes **L** and **NL**, we choose the logspace reduction for RR-problems. We say that filter F_1 dominates filter F_2 if $\text{RR}(F_2) \leq_{\log} \text{RR}(F_1)$. The natural goal is to describe dominance relation on filters by some structural properties of languages. Even finding relation on filters, that respects dominance relation is a hard problem: we know only one such relation – deterministic finite state transduction. We define this relation as the relation provided by deterministic finite state transducer. First we recall the definition of finite state transducer, which is also known as rational transducer.

Formally a finite state transducer is defined by tuple $T = (A, B, Q, q_0, \delta, F)$, where A is the input alphabet, B is the output alphabet, Q is the (finite) state set, q_0 is the initial state, $F \subseteq Q$ is the set of accepting states and $\delta: Q \times (A \cup \varepsilon) \times (B \cup \varepsilon) \times Q$ is the transition relation.

As in case of automata, we say finite state transducer to be deterministic if transition relation δ is a function.

Consider two DFSTs T_1 and T_2 . We say that a DFST $T = T_1 \circ T_2$ is the composition of T_1 and T_2 if $T(x) = y$ iff $T_1(x) = u$ and $T_2(u) = y$.

Define the composition of transducer T and automaton \mathcal{A} in the same way: we say that automaton $\mathcal{B} = T \circ \mathcal{A}$ recognizes language $\{x \mid T(x) = y \in L(\mathcal{A})\}$.

The following proposition is an algorithmic version of Elgot-Mezei theorem (see, e.g., [1, Th. 4.4]).

Proposition 1. *The composition of transducers and the composition of a transducer and an automaton are computable in deterministic log space.*

We say that filter F_1 covers filter F_2 if there exists such dfst T , that $F_2 = T(F_1)$. We also write this as $F_2 \leq_{\text{dfst}} F_1$.

Lemma 1. *If $F_1 \leq_{\text{dfst}} F_2$ then $\text{RR}(F_1) \leq_{\log} \text{RR}(F_2)$.*

Proof. Let T be a deterministic finite state transducer such that $F_1 = T(F_2)$ and let \mathcal{A} be an input of the $\text{RR}(F_1)$ problem. Build the automaton $\mathcal{B} = T \circ \mathcal{A}$ and use it as an input of the $\text{RR}(F_2)$ problem. It gives the log space reduction due to Proposition 1.

We obtain similar results on relation between RR^n problems and rational (not necessary deterministic) transductions in the paper about RR -problems and context-free languages [2].

Now we divide the class of regular languages into two parts. We call regular filter F *hard* if F covers an arbitrary regular language. It means that for every regular language R there exists DFST T , such that $R = T(F)$; in the other case we call regular filter F *easy*.

Proposition 2. *Regular language F is hard iff an arbitrary deterministic automaton, recognizing F , has paths $q \xrightarrow{u} q$, $q \xrightarrow{v} q$, for some state q and some words $u \neq vw, v \neq uw$.*

Proof. Consider automaton \mathcal{A} , recognizing F , such that from each state s there is a path to some accepting state. By the condition of the proposition there is some state q and words u, v , such that $q \xrightarrow{u} q$, $q \xrightarrow{v} q$, $u \neq v$, moreover u is not a prefix of v and v is not a prefix of u . Let p be the path $q_0 \xrightarrow{p} q$ from the initial state to state q and s be the pass $q \xrightarrow{s} q_f$ from state q to some accepting state q_f .

First we build DFST T , which maps F to Σ^* . Assume, without loss of generality, that we work with the binary alphabet $\Sigma = \{a, b\}$. Describe the construction of T . First transducer T expects the word p on the input, processes it and writes nothing. Then if T reads word u , it writes letter a , if T reads word v , it writes letter b . Since u is not a prefix of v and v is not a prefix of u , transducer T writes a and b independently. If T reads word s it goes to the (only) accepting state.

It is clear that $T(F) = \Sigma^*$. Now it is easy to build DFST T_R , which maps Σ^* to regular language R and to build the composition $T \circ T_R$ as a resulting transducer T' , which maps F on R . To build T_R we take an arbitrary DFA \mathcal{A} , recognizing R and turn it to DFST T_R by adding output tape and modifying transition function by adding to output the letter of transition: if the automaton has transition $\delta_{\mathcal{A}}(q, a) = q'$, then the transducer has transition $(q, a, a, q') \in \delta_{T_R}$.

Let us call cycle the path of form $q \xrightarrow{u} q$. Now we prove that if there exists an automaton, recognizing F , without distinct cycles, then there is no transducer T , that maps F to Σ^* . Notice that in this case language F can be described by regular expression consisting of finite union of expressions of form $px_1^*y_1x_2^*y_2 \dots x_n^*y_ns$ – since there are no two distinct cycles, if state q_i has some nonempty cycle, then there is word x_i , such that each path $q_i \xrightarrow{w} q_i$ can be described as $w = x_i^k$. It is easy to see, that in case of one expression of form px^*s , language F doesn't cover Σ^* . Indeed, if there is a transducer T , such that $T(px^*s) = \Sigma^*$ then for long enough word w from Σ^* should exist a long enough word from px^*s . But since transducer T has finitely many states there are such numbers n and k , that $q_0 \xrightarrow{px^n} q$ and $q_0 \xrightarrow{px^{n+k}} q$, so we get that each long

enough word from $T(px^*s)$ has a periodic subword, and come to contradiction – there are words without periodic factors. Using the pigeon-hole principle again we obtain, the same contradiction in general case: for the expressions of form $px_1^*y_1x_2^*y_2\ldots x_n^*y_ns$ and thus we obtain the same contradiction for their finite union. Finite union is contained in some regular language of form $w_1^*w_2^*\ldots w_n^*$ and since that language does not cover Σ^* , finite union also doesn't.

Recall, that regular language R is called *bounded* if there are such words w_1, \ldots, w_n , that $R \subseteq w_1^*w_2^*\ldots w_n^*$. From the proof of proposition 2 we obtain the corollary.

Corollary 1. *Regular language is easy iff it is a bounded regular language.*

Remark 1. If regular language F is easy, then $F \leq_{\text{dfst}} w_1^*w_2^*\ldots w_n^*$.

Proof. Indeed, since $F \subseteq w_1^*w_2^*\ldots w_n^*$, we just apply DFST ID_F (which maps w to w iff $w \in F$) to the language $w_1^*w_2^*\ldots w_n^*$.

3 Generalized Nondeterministic Automata

Regular realizability problems have corresponding computational model, called Generalized Nondeterministic Automata. By *generalized nondeterministic automaton* M_F (depending on filter F) we mean deterministic logspace Turing machine with advanced one-way read-only tape. GNA M_F accepts word w if there exists such word $adv \in F$, that M accepts w when adv is written on the advanced tape. We call the advanced tape *advice tape*.

Let us describe the model a little more formally. Advice tape has an alphabet Δ , and blank-symbol $\Lambda \notin \Delta$. We define $M_F(w, \alpha)$ to be the function of two arguments – word w on the input tape and word $\alpha \in F \subseteq \Delta^*$ on the advice tape. After word α advice tape is filled with blank-symbols Λ . On each step GNA can move the head of advice tape and read the next symbol or not to move the head. We assume, that filter is always a non-empty language. So, the function $M_F(w, \alpha)$ equals 1 if GNA M_F reaches some accepting configuration on pair (w, α) and in the other case we assume that $M_F(w, \alpha) = 0$ (we assume, that GNA stops on each pair). We say that GNA M_F accepts word w if there is such advice $\alpha \in F$, that $M_F(w, \alpha) = 1$ and as usual by language $L(M_F)$ we mean all the words, accepted by M_F .

Why do we use the word automata? First, GNA was defined in [3] as a multi-head two-way automata with an additional read-only tape, but the equivalent model appeared to be more useful in proofs. The equivalence between multi-head two-way automata and logspace machines was proofed by Cobham in his unpublished paper as we know from [4].

Each regular realizability problem $\text{RR}(F)$ has the corresponding GNA M_F , by corresponding we mean, that $\mathcal{A} \in \text{RR}(F)$ iff $\mathcal{A} \in L(M_F)$.

Theorem 1. [5]
 $\text{RR}(F) \leq_{\log} L(M_F)$.

And there is also a reduction in the other direction.

Theorem 2. [5]

$$L(M_F) \leq_{\log} \text{RR}(F).$$

If the filter contains only empty word, than GNA $M_{\{\varepsilon\}}$ turns to a deterministic logspace machine. In case when filter is the language of all words (under binary alphabet) M_{Δ^*} turns to standard nondeterministic logspace machine. The intermediate case is the object of exploring of regular realizability problems.

4 Main result

Recall, that we call regular filter F *hard* if for an arbitrary regular language R there exists DFST T , such that $R = T(F)$; in the other case we call regular filter F *easy*.

Lemma 2. *If regular filter F is hard, then the problem $\text{RR}(F)$ is **NL**-complete.*

Proof. The statement is obvious in case $F = \Delta^*$ – the definition of GNA turns to the definition of nondeterministic logspace machine. In other case consider DFST T , such that $\Delta^* = T(F)$ and we obtain the reduction $\text{RR}(\Delta^*) \leq_{\log} \text{RR}(F)$ by lemma 1. So $\text{RR}(F)$ is **NL**-hard. We use another DFST $T' : F = T'(\Delta^*)$ for reduction in the other direction.

Lemma 3. *If regular filter F is easy, then $\text{RR}(F) \in L$.*

Proof. Recall, that all easy regular languages are bounded and there for $F \leq_{\text{dfst}} u_1^* u_2^* \dots u_k^*$ by remark 1. That's why we only prove the lemma in case of languages of form $u_1^* u_2^* \dots u_k^*$. For GNA M_F we built an equivalent logspace machine M . We equip M with k counters, and program it to recursively try advices $u_1^{i_1} u_2^{i_2} \dots u_k^{i_k}$. For each tuple of indices there is only finite number of different configurations of M_F on input w so since the number of configurations is polynomial of $|w|$, then each index-counter can also be bounded by polynomial of $|w|$, then machine M can try all possible reasonable advices and verify whether GNA M_F accepts word w .

Theorem 3. *If regular language F is hard, then the problem $\text{RR}(F)$ is **NL**-complete; if regular language F is easy, then the problem $\text{RR}(F)$ belongs to class L .*

Now we compare RR and RR^n problems.

Proposition 1. *For each non-empty regular filter F nondeterministic regular realizability problem $\text{RR}^n(F)$ is **NL**-complete.*

Proof. First we prove that $\text{RR}^n(F) \in \mathbf{NL}$. Let NFA \mathcal{A} be an input of the problem and \mathcal{B} be an NFA recognizing F . The NL-algorithm nondeterministically guesses paths from initial to final states in automata \mathcal{A} and \mathcal{B} and verifies that both paths correspond to the same word. Now we prove that each nonempty language L is \mathbf{NL} -hard. Let $w \in L$. We reduce the path problem to $\text{RR}^n(L)$. Let $G(s, t)$ be an input of path problem. We built NFA \mathcal{A} from graph G by writing ε on each edge, converting s to initial state and adding path labeled by w from vertex t to the only accepting state. So $w \in L(\mathcal{A})$ iff there is a path from s to t in graph G .

Proposition 3. *If regular filter F is easy then $\text{RR}(F) \not\sim_{\log} \text{RR}^n(F)$ modulo $L \neq \mathbf{NL}$.*

References

1. J. Berstel. Transductions and context-free languages. Teubner Verlag, 1979.
2. Rubtsov A.A., Vyalys M.N. Regular realizability problems and context-free languages, eprint arXiv:1503.00295.
3. Vyalys M.N. On Nondeterminism Models for Two-Way Automata, in Proc. VIII Int. Conf. on Discrete Models in Control System Theory, Moscow, 2009, Moscow: MAKS Press, 2009, pp. 5460.
4. Ibarra O.H. Characterizations of Some Tape and Time Complexity Classes of Turing Machines of Multihead and Auxiliary Stack Automata. J. of Comp. and Sys. Sci. 5, 88–117 (1971)
5. Vyalys M.N. On regular realizability problems. Problems of Information Transmission. Vol. 47, issue 4, 2011. P. 342–352.